Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis

Randal E. Bryant*
Fujitsu Laboratories, Ltd.
Kawasaki, JAPAN

Abstract

The program TRANALYZE generates a gate-level representation of an MOS transistor circuit. The resulting model contains only four-valued unit and zero delay logic primitives, suitable for evaluation by conventional gate-level simulators and hardware simulation accelerators. TRANALYZE has the same generality and accuracy as switch-level simulation, generating models for a wide range of technologies and design styles, while expressing the detailed effects of bidirectional transistors, stored charge, and multiple signal strengths. It produces models with size comparable to ones generated by hand.

1 Introduction

Switch-level simulation has proved an effective means for verifying digital circuits implemented in MOS technology. By directly working from a transistor representation, a switch-level simulator can handle a large range of circuit designs and capture such subtle effects as bidirectional signal flow, dynamically stored charge, and multiple signal sources with different driving impedances. Traditionally, switch-level simulation requires evaluation mechanisms that are not found in conventional gate-level simulators. To utilize the features and modeling libraries of existing simulators, and for execution on gate-level hardware simulation accelerators, we would like to overcome this incompatibility.

By automatically generating gate-level models from transistor circuits, we can provide a simulation methodology that combines switch-level generality and accuracy with gate-level compatibility and performance. In taking this approach, we should take care to satisfy several design constraints. First, we must not give up the generality and accuracy of switch-level simulation. Second, the generated models should be suitable for evaluation by any gate-level simulator. Third, the generated models should be as compact as possible.

1.1 Previous Work

A variety of programs have been developed that fall into the general category of gate-level model extractors. On close examination, however, we see that all of them fall short in at least one of the design goals listed above.

*This paper appeared in the 1991 *International Conference on Computer-Aided Design (ICCAD '91)*. On leave from School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

Most model extractors operate by repeatedly applying transformation rules that eliminate or reduce some of the transistor logic and replace it by logic gates [3, 4]. These rules typically include transformations such as: series/parallel reduction of pullup and pulldown networks; recognition of static logic gates; identification of complementary control signals; merging transistor pairs in transmission gates; elimination of "uninteresting" nodes [1]; transistor direction assignment [2]; and recognition of special structures such as multiplexors and busses. Often, these programs encounter cases where none of the transformation rules apply, but transistors remain in the circuit model. The user must then either supply hand-generated models or add new transformation rules to the model extractor.

An alternative to rule-based approaches is to generate models using methods derived from switch-level simulation. This has the advantage of starting from a firm foundation in terms of generality and accuracy. The challenge becomes to satisfy the remaining goals of compatibility and conciseness. Symbolic switch-level analysis, as exemplified by the ANAMOS program [7] (the preprocessor for the COSMOS simulator [5]), can extract a logic description from the transistor network. ANAMOS generates models consisting of Boolean DAGs: networks of 2-valued operations describing the computation for one unit time step of the simulator. These DAGs operate on encoded signal values—each 3-valued node state is encoded as a pair of binary values.

In earlier work at CMU, we developed a program HLGCC to convert the Boolean DAGs generated by ANAMOS into logic gates [9]. HLGCC reduces the model size by mapping the two-valued operations of the Boolean DAGs into 3-valued logic primitives, and merging multiple operations into three-input logic gates. These optimizations, however, had only limited success. As an example, for a CMOS transmission gate multiplexor, ANAMOS generates a DAG containing 17 Boolean operations, which HLGCC maps into 10 three-input gates. Ideally, we should be able to generate a single gate model for this circuit. Several reasons can be identified for this shortcoming. First, by operating on encoded signal values, ANAMOS loses much of the original context about the node state values. Second, by analyzing each channel-connected component independently, ANAMOS cannot exploit correlations between signals, e.g., complementary signal pairs. Finally, ANAMOS generates a very conservative model, assuming that all transistors have unit delay, and that the effects of `X` signals should be modeled very strictly. Thus, the generated models must include many terms for evaluating sneak paths and unit delay glitches.

1.2 New approach

As with ANAMOS, TRANALYZE performs a symbolic, switch-level analysis to extract a logic representation of the circuit behavior. Instead of using a binary encoding of signal values, however, it performs the analysis in an algebra with values `0`, `1`, `X`, and `Z`. The fourth value `Z` is similar to the "high impedance" value used by many gate-level simulators for simulating tri-state drivers and busses. The analysis algorithm has the same general form as that used in ANAMOS [7]—each channel-connected component is analyzed by setting up systems of equations, which are solved by a symbolic form of Gaussian elimination [6] to yield the logic representation. TRANALYZE differs from its predecessor in the following key respects: it represents and manipulates its logic description as a 4-valued gate-level network rather than as a Boolean DAG; it analyzes the channel-connected components in rank order, so that it can exploit the correlations between the input signals of each component; and the optimizer can optionally generate models with less conservative, and hence simpler handling of `X` signals.

During the analysis, TRANALYZE applies extensive optimizations of the gate-level model to reduce its size. The combination of symbolic analysis plus logic optimization effectively performs many of the circuit optimizations implemented explicitly by rule-based systems.

2 Generated gate-level models

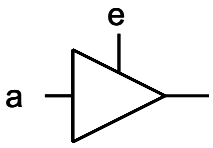
In a switch-level network, each node may have state 0, 1, or X, where X indicates either an unknown value or a potentially nondigital voltage. To represent intermediate terms in the analysis, we introduce a fourth value Z indicating the absence of a signal. In the gate-level model produced, we can guarantee that no circuit node will ever have state Z—at the very least an isolated node will retain its stored charge.

An MOS transistor can serve in two different capacities. First, as in relay contact networks, series-parallel (and other) configurations can be formed that conditionally connect two terminals according to some logic function of the transistor gate nodes. We will refer to this as "And/Or" logic. Second, a transistor can propagate a data value from its source to its drain (or vice-versa) under the control of its gate node. We will refer to this as "Steering" logic. Typical MOS circuits contains both forms of logic. Logic gates are created using And/Or logic as pullup and pulldown networks. Transmission gates, multiplexors, and carry chains are created using steering logic. Our signal algebra includes operators to express both forms of logic.

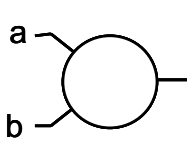
2.1 Primitive gates

We express And/Or logic using gate types AND and INVERT. The OR operation is expressed in terms of these two gate types according to DeMorgan's Laws, aiding the detection of identical and complementary terms. We can guarantee that the Z state will never arise in And/Or logic.

We express Steering logic using gate types ENABLE and MERGE. These operations are defined as follows, where entry '–' indicates a condition that will never arise in the generated gate-level network:



Enable	e = Z	e = 0	e = 1	e = X
a = Z	–	Z	Z	Z
a = 0	–	Z	0	0
a = 1	–	Z	1	1
a = X	–	Z	X	X



Merge	a = Z	a = 0	a = 1	a = X
b = Z	Z	0	1	X
b = 0	0	0	X	X
b = 1	1	X	1	X
b = X	X	X	X	X

The ENABLE gate conditionally propagates the data on input a according to the control signal on input e. As the table indicates, the control input can never equal Z—its value is generated by And/Or logic. This gate has functionality similar to that of a tri-state buffer, except that it treats control signal value X the same as a 1. TRANALYZE resolves the effects of unknown control signals on tri-state buffers by including additional gates in the model. The MERGE gate combines two 4-valued signals. Its functionality is identical to a "wired-logic" function in tri-state logic. As a final gate type, the DELAY gate implements a unit delay. All other gates have zero delay.

2.2 Gate-level logic example

Figure 1 shows a CMOS circuit and the generated gate network. As can be seen, the program successfully recognizes that the pullup and pulldown networks form a NOR gate driving node S. The logic generated for node T selects either the output of the NOR gate, or the stored value on node T (delayed by 1 time unit), depending on control signal C. After technology mapping, TRANALYZE generates a 2 gate model, consisting of a NOR and a multiplexor. In contrast, HLGCC generates a 7 gate model.

3 Analysis method

Following the parsing of a transistor netlist file, the analysis proceeds by a series of steps.

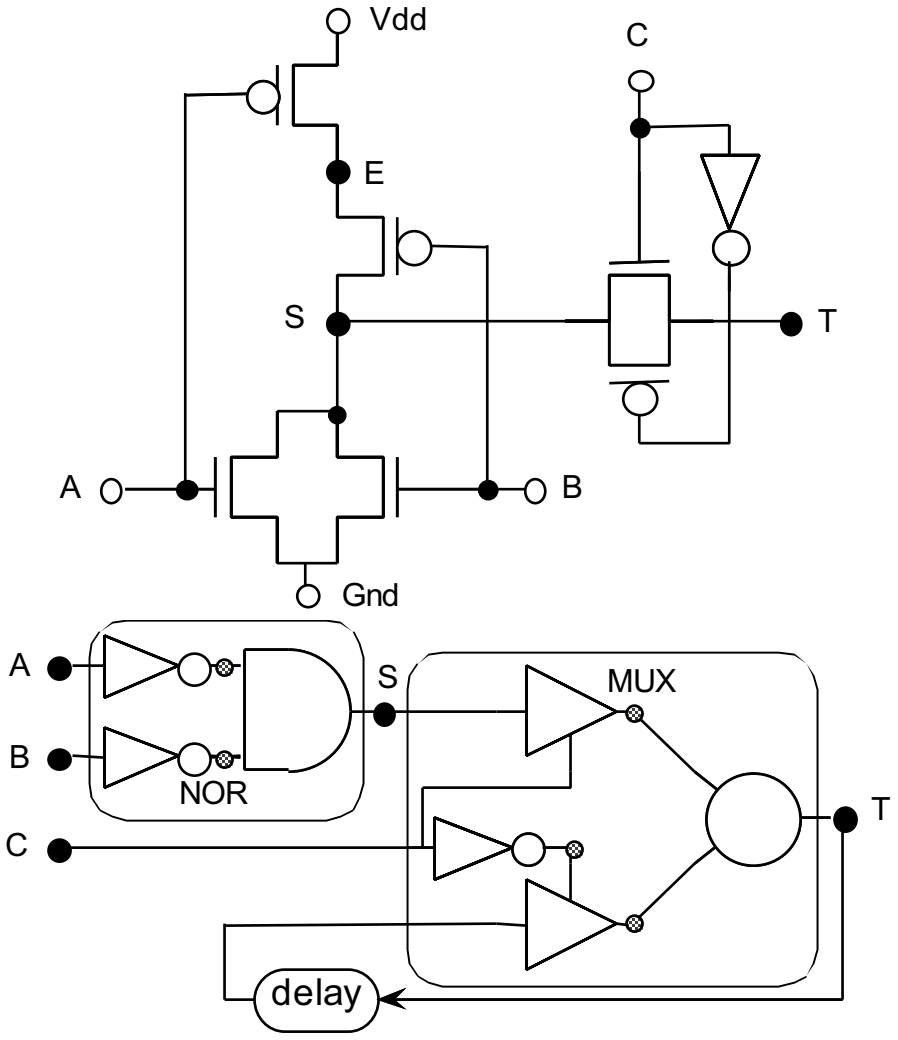


Figure 1: **Example Circuit and Generated Model**

3.1 Partitioning and ordering

The transistor circuit is first partitioned into "channel-connected components," each consisting of a set of storage nodes connected by the source-drain terminals of transistors. Each such region is analyzed separately.

TRANALYZE exploits the logical dependencies implied by zero delay transistor logic in its logic optimization. It does this by analyzing the circuit components in *rank order*, so that as a component is analyzed, the gate-level models for the signals controlling zero delay transistors are available to the optimizer. TRANALYZE automatically inserts unit delays on some of the transistors to enable a rank ordering and to avoid generating a gate network with zero delay cycles. It does this by a simple greedy method during the topological sorting of the components.

3.2 Symbolic analysis

During the analysis of a channel-connected component, gates are added to the generated network describing the functionality of the component nodes. Each component node is temporarily treated as a primary input fed through a unit delay to represent the initial node charge. Working from the maximum strength level downward, two systems of equations are set up and solved symbolically at each strength level. The gate outputs representing the final steady state solution are then connected to the primary inputs that were temporarily introduced for the storage nodes.

For strength level s, the first system of equations, termed the "clear" equations, expresses the conditions under which each node is not the destination of a definite path of strength s. These equations are expressed in terms of And/Or logic. The second system, termed the "state" equations, expresses the combined effect of all unblocked paths of strength greater than or equal to s to each node. These equations are expressed in terms of Steering logic, with ENABLE expressing conditional signal propagation and MERGE expressing the effect of multiple signals to a single destination.

3.3 Symbolic manipulation

During the setting up and the solving of equations, each signal corresponds to either a primary input or the output of a logic gate. To "compute" the result of applying some operation (AND, MERGE, etc.) to a set of argument signals, the symbolic manipulator either finds an existing gate with the appropriate functionality, or it adds a new gate to the network having the argument signals as inputs. The symbolic manipulator also applies extensive optimizations as it proceeds. Most of the optimizations are similar to those found in ANAMOS, as well as in many optimizing compilers, e.g., constant evaluation, common subexpression detection, etc., except that it performs these optimizations in the 4-valued signal algebra. Although most of these optimizations require only constant time, others involve attempting a simple form of proof by contradiction to show that a candidate gate would always produce an output value equal to one of its inputs.

Unlike ANAMOS where the effects of X signal values are modeled very conservatively, TRANALYZE can generate models with "Boolean" optimization, ignoring the effects of X values in And/Or logic. For most applications, designers are willing to give up detailed modeling of X values in favor of simpler gate-level models. Indeed, they would find the gate model generated by Boolean optimization less prone to false X value propagation.

Additional transformations are implemented by the symbolic manipulator to convert Steering logic into And/Or logic. The first two illustrated in Figure 2 effectively perform series-parallel reduction of the transistor network. The first takes a chain of ENABLE gates, such as arises from the analysis of a series transistor chain, and collapses it into a single ENABLE controlled by an AND. The second takes the MERGE of a set

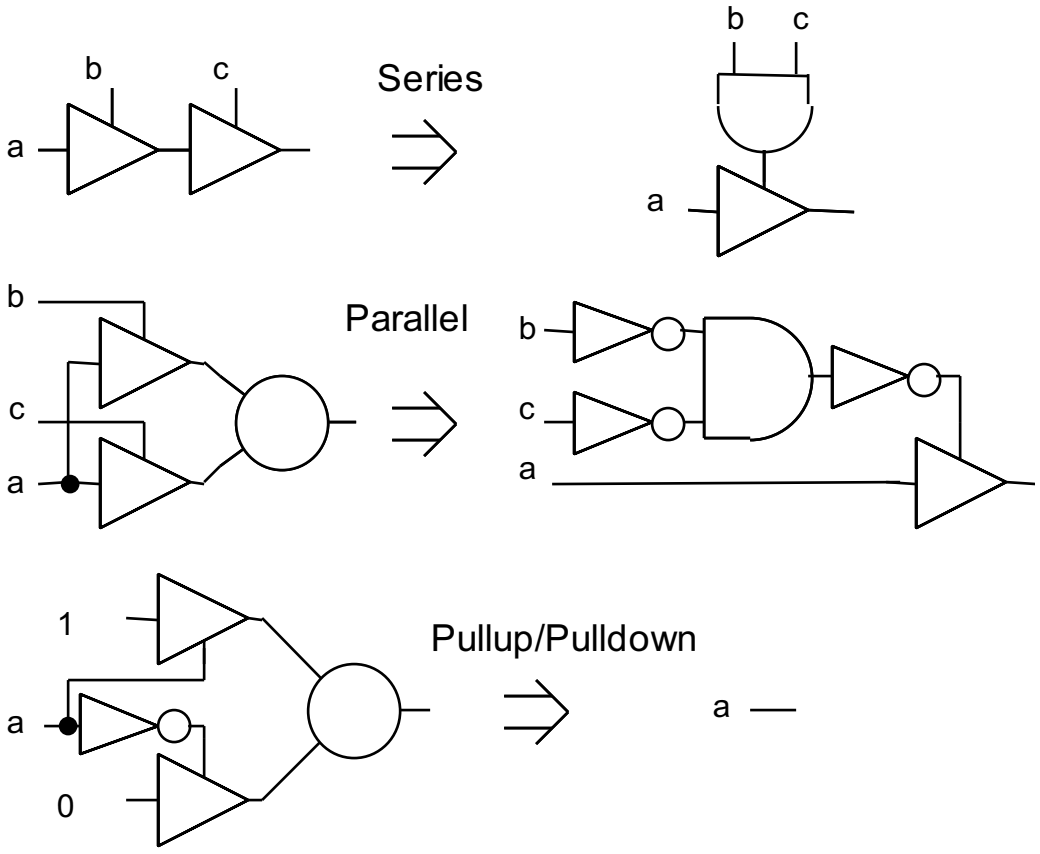


Figure 2: **Example Transformation Rules**

Circuit	**Trans.**	ANAMOS	HLGCC		TRANALYZE (Unit/Ternary)			TRANALYZE (Zero/Binary)		
		2 inp.	2 inp.	3 inp.	2 inp.	3 inp.	4 inp.	2 inp.	3 inp.	4 inp.
74181 ALU	240	265	193	131	194	120	98	185	122	89
Figure of Merit		**1.10**	**0.80**	**0.55**	**0.81**	**0.50**	**0.41**	**0.77**	**0.51**	**0.37**
Shift64	658	1669			773	454	326	515	323	195
Figure of Merit		**2.54**			**1.17**	**0.69**	**0.50**	**0.78**	**0.49**	**0.30**
DRAM256	1140	8346	8562	4888	5430	3172	2361	5296	3087	2310
Figure of Merit		**7.32**	**7.51**	**4.29**	**4.76**	**2.78**	**2.07**	**4.65**	**2.71**	**2.03**
SLAP	20167	80057	74498	45085	67035			42861		
Figure of Merit		**3.97**	**3.69**	**2.24**	**3.32**			**2.13**		

Figure of Merit: (Gate Count) / (Transistor Count)

of ENABLE gates having a common data input, such as arises from the analysis of a set of parallel paths, and collapses it into a single ENABLE gate controlled by the OR of the parallel control signals. This OR is implemented as a combination of INVERT and AND to exploit the equivalences implied by DeMorgan's Laws. The third rule of Figure 2 illustrates the final transformation required to extract static logic gates. The reduction of the pullup and pulldown networks by series-parallel transformations yields a pair of complementary signals. These signals control ENABLE gates to constants 1 and 0, representing power and ground. This final configuration of ENABLE and MERGE gates can then be eliminated.

3.4 Network pruning

During the analysis, TRANALYZE generates gate level logic describing the functionality of every storage node in the circuit, including such nodes as the intermediate points in series transistor chains. TRANALYZE prunes the network by a form of mark-sweep garbage collection. Starting with the gates representing the circuit nodes that the user wishes to observe during simulation, the program traces backward, marking all reachable gates. Any unmarked gate can then be eliminated; its output value can have no bearing on the simulation results. In this manner, the program prunes a large fraction of the nodes from the transistor circuit, eliminating the need for heuristic methods to identify these nodes initially [1]. As an example, connection node E in the pullup chain of the NOR gate of Figure 1 is successfully eliminated by the pruning.

3.5 Technology mapping

As the final stage, the primitive gates are merged into a smaller number of more complex gate types. The initial target simulator for TRANALYZE is the hardware simulation machine SP [8]. This machine can model arbitrary 4-valued gates with up to four inputs. Our technology mapper merges trees of gates, using a simple tree matching algorithm [10].

4 Experimental results

The table above indicates the results of the different analysis methods for several switch-level benchmarks. Results are given for 4 different representations: the Boolean DAG model generated by ANAMOS, the ternary gate model generated by HLGCC, and the 4-valued models generated by TRANALYZE for two extremes of network optimization. Unit/ternary indicates that all transistors have unit delay, and X values are modeled conservatively. The resulting model is functionally equivalent to that produced by ANAMOS. Zero/binary indicates that transistors are assigned unit delay only to break feedback loops, and with Boolean optimization.

As a figure of merit, ratios are given of the number of gates (or DAG nodes) to the number of transistors in the circuit. Low (less than 1.0) values indicate that the program is able to abstract the circuit behavior. High numbers indicate cases where a complex gate-level model is required to capture the subtleties of switch-level behavior. As this table indicates, TRANALYZE consistently outperforms both ANAMOS and HLGCC, even when forced to generate models with exactly the same functionality.

The 74181 ALU is a direct mapping of a gate-level ALU into static CMOS gates. Both HLGCC and TRANALYZE successfully recognize all of the gates, except for the XORs. The Shift64 circuit is a 64-bit transmission gate shift register that can either shift or hold its data on each clock cycle. TRANALYZE can reduce each stage to just 3 four-input gates—comparable to a hand generated model. The DRAM circuit is an nMOS 3-transistor dynamic RAM. This represents a difficult case for gate-level model generation, since any gate-level implementation of a RAM is far more complex than what can be implemented using custom transistor logic. The SLAP circuit is a 16-bit CMOS processor designed at CMU. It contains many difficult structures for gate-level model extraction, including a register file implemented as a static RAM, a Manchester carry chain ALU, and a transmission gate shifter network. Even for these more difficult circuits, TRANALYZE is able to generate reasonably concise models.

Thus far, we have successfully simulated all of the benchmark circuits except for SLAP on SP. For the largest circuit simulated (DRAM256), SP operates 80 times faster than COSMOS executing on a SUN-4/110. Even greater speedups can be expected for larger circuits.

Acknowledgements

Both S. Shimogori and M. Shoji of Fujitsu Laboratories have been instrumental in mapping the output of TRANALYZE onto SP, and in executing the benchmark simulations.

References

[1] D. T. Blaauw, P. Banerjee, and J. A. Abraham, “Automatic classification of node types in switch-level descriptions,” *ICCAD*, 1990.

[2] D. T. Blaauw, D. G. Saab, J. Long, and J. A. Abraham, “Derivation of signal flow for switch-Level simulation,” *EDAC*, 1990, 301-305

[3] D. T. Blaauw, D. G. Saab, P. Banerjee, and J. A. Abraham, “Functional abstraction of logic gates for switch-level simulation,” *EDAC*, 1991.

[4] M. Boehner, “LOGEX—An automatic logic extractor from transistor to gate level for CMOS technology,” *25th DAC*, 1988, 517–522.

[5] R. E. Bryant, *et al*, “COSMOS: a compiled simulator for MOS circuits,” *24th DAC*, 1987, 9–16.

[6] R. E. Bryant, “Algorithmic aspects of symbolic switch network analysis,” *IEEE Trans. CAD/IC*, 1987, 618–633.

[7] R. E. Bryant, “Boolean analysis of MOS circuits,” *IEEE Trans. CAD/IC*, 1987, 634–649.

[8] F. Hirose, K. Takayama, and J. Niitsuma, “An event-driven logic simulation machine of computer systems,” *Proc. 1990 European Simulation Multiconference*, Nuremburg, Germany, June, 1990.

[9] A. Jain, and R. E. Bryant, “Mapping switch-level simulation onto gate-level hardware accelerators,” *28th DAC*, 1991.

[10] K. Keutzer, “DAGON: Technology binding and local optimization by DAG matching,” *24th DAC*, 1987, 341–347.